Trust Information Delivery Scheme for Certificate Validation

BACKGROUND OF THE INVENTION

TECHNICAL FIELD

The invention relates to maintaining security in an electronic network. More particularly, the invention relates to a trust information delivery scheme for certificate validation.

DESCRIPTION OF THE PRIOR ART

The Secure Socket Layer (SSL) protocol (see The SSL Protocol Version 3.0, http://www.netscape.com/eng/ssl3/ssl-toc.html) is presently the *de facto* industry standard for Web security. In fact, because most E-commerce applications are based on the Web, the SSL protocol is built into almost all Web servers and browsers, such as Netscape Enterprise Server, Microsoft Web Server, Netscape Communicator, and Microsoft Internet Explorer (IE).

The SSL protocol uses public key cryptography in conjunction with an X.509 certificate to provide server authentication and, optionally, client authentication. During the authentication process, the server sends its X.509 certificate chain which may or may not contain the root CA certificate who signed the server certificate to the client as part of the handshake messages the client and server exchange at the start of a session. The client validates the server's certificate through a normal certificate verification procedure if it

1

has the server's certificate, it has the certification authority's (CA) certificate that signed the server's certificate, and it has associated trust information.

A common approach for providing the CA certificates and associated trust information to the client is to hard code a certificate database into the client software, such as done in Netscape Communicator and Microsoft IE. In this scheme, certificate management is up to the end users. For example, if a CA certificate is expired, it is the end users' responsibility to perform a new CA certificate rollover. Furthermore, the client hardware needs sufficient storage space to hold the certificate database, which usually ranges from 100K to 400k. This is typically not a problem in the personal computer environment.

However, the above approach can not be applied to mass market applications that run on the consumer appliances, such as set-top boxes, hand-held computers, pagers, and cell phones, due to at least any of the following reasons:

• Public CA root certificates, as with substantially all certificates, expire and their lifetime may not fit nicely within the expected shelf-life or lifetime of a consumer product. Thus, a mechanism for securely updating CA root certificates is desirable.

2. In the unlikely event that a public CA's private root key is compromised, its root certificate must be revoked. Some trusted entity has to assume responsibility for revoking all compromised CA root certificates. In the personal computer environment, end users can revoke the root certificate

2

by updating the client software if the compromised CA certificate is embedded in the client software. However, updating the software per each expired certificate is not practical in the consumer product environment because, *e.g.* it is practically impossible to recall all of the devices for

5   participation in the update.

2. The typical consumer electronics user is unsophisticated with regard to security and, as such, cannot be expected to participate in the maintenance of the trust knowledge information base. In the consumer

10   device environment, maintenance of the trust knowledge information base must occur transparently to the user and in a secure fashion.

2. Most consumer electronic devices have a limited amount of non-volatile flash memory where CA root certificates and associated trust information

15   can be stored. It is important to minimize the usage of this memory space.

Current or Known Solutions

The SSL protocol itself does not specify how the root CA certificate validation

20   should be implemented. Thus, such implementation is very much vendor dependent and proprietary to each vendor. There are very few publications regarding to this issue. Some well-known implementations are Netscape Communicator and Microsoft IE, which are based on a certificate database embedded in the browser software. In March 2001, an IETF draft, the Simple

25   Certificate Validation Protocol (SCVP; see Simple Certificate Validation

Protocol, http://search.ietf.org/internet-drafts/draft-ietf-pkix-scvp-06.txt) regarding such certificate validation was published. The proposed protocol is based on a server that performs certificate validation.

5   The SCVP scheme uses a simple request-response model, which is briefly described as follows:

Upon receiving a certificate, the client sends a certificate validation request to a server. The request is optionally signed and contains the certificate in question and other parameters, such as extensions, as well as an answer that
10   the client wants the server to provide. Based on the request, the server generates a response, signs it, and returns it to the client.

One advantage of the mechanism provided in this proposal is that it simplifies
15   the client implementation for certificate validation, and it can be used as a general approach for clients to verify received certificates, including server certificates, in SSL. However, the protocol only addresses communications and does not address certificate management at the client endpoint. For example, it is important to provide a mechanism for delivering and updating
20   root certificates that are needed to verify the server's responses, but such mechanism is not addressed in the proposal. The set of root certificates is the initial trust point for such protocol. Without an initial trust point, the certificate validation mechanism is still vulnerable.

25   Liberate Technologies of San Carlos, California has developed a server-based certificate management mechanism that securely delivers root

certificates and associated trust information to clients. With this scheme, the server certificates in SSL can be seamlessly validated. Such scheme fits particularly well into consumer devices and solves the problem of maintaining an initial trust point in the certificate chain, discussed above.

5

The Liberate mechanism works substantially as follows:

The building block of the scheme is the Root Security Information Object (RSIO), which is a data object that contains the following sets of information:

10

- A chain of Trusted Information Provider's (software provider, device owner, and Enhanced Service Provider (ESP)) Root Certificates. For the device owner and ESP, the chain only contains the most recent device owner/ESP root certificate.

15

- A trust table consisting of a set of SHA-1 hash values of the trusted certificates. For each hash value in the table, there is a bit vector describing the operations for which the certificate is trusted, *i.e.* the trust vector, and a bit vector describing the operations which the certificate is
20 trusted to delegate to other certificates. The table is digitally signed by the subject who provided the object.

- A timestamp indicating the time this object is created.

25 • The digital signature of the trust table.

The Liberate mechanism typically involves three business entities, *e.g.* the software provider, a device owner, and an ESP, which forms a hierarchy with the software provider at the top, the device owner in the middle, and the ESP in the bottom. Each of the three entities provides an RSIO that is based on its own requirements. The root certificate of each entity must be submitted to, and verified by, the entity in the next higher level of the hierarchy and included in its RSIO. Thus, the RSIOs also form a hierarchy linked by their root certificates with the software provider RSIO at the top, the device owner RSIO in the middle, and the ESP RSIO at the bottom. Based on the RSIOs, the server constructs a Hierarchical Root Security Information Object (HRSIO) and delivers it to the client during the client boot up time.

The HRSIO contains the following information:

- A chain of software provider Root Certificates.

- A trust table signed by the most recent software provider Root Certificate.

- One or more trust tables signed by trusted device owners and the device owner's Root Certificates.

- A trust table signed by a trusted ESP and the ESP Root Certificate.

6

The software provider root certificate is stored into the ROM of each client device during the manufacture process. Thus, upon receiving a HRSIO, the client can verify it through a chain of validation.

5   When a client attempts to initiate an SSL connection to a server, it proceeds through the phases of the SSL handshake. The discussion herein does not attempt to present the SSL protocol in detail, but rather highlights the points at which HRSIO information is used. In the initial phases of the handshake, the server presents its certificate and, optionally, the CA certificate that signed its

10   certificate.

The proposed scheme has a significant advantage over the old scheme because of the following:

> In the RSIO scheme, each hash value in the trust table is computed using the whole certificate information as the input.
> Consequently, having two certificates with different validity (a field contained in the certificate) results in two different hash values. This property creates an unnecessary failure in the SSL certificate chain validation in a common scenario:

20

When the client receives a server certificate chain that contains the renewal of an expired root CA certificate and the client trust table only contains the old root CA certificate because the trust table has not been updated yet. In this case, the SSL certificate chain validation fails because the hashes of the two

25   certificates do not match. That means, whenever a certificate expired in the RSIO, the corresponding RSIO provider has to update the certificate in its RSIO and reconstruct a new RSIO and deliver the updated trust table to the client. This actually needs to go through the whole RSIO update procedure.

30   However, it is a common practice to renew the CA certificate by extending the validity of the old certificate based on the same key pair if it has not been compromised. This practice is to minimize migration problems for the CAs.

7

Thus, in this case, the RSIO update is unnecessary because the old certificate still should be treated as valid even it is expired because the old and the new certificate share the same key. This problem is perfectly solved in the proposed scheme by hashing the public key portion of the certificate during the creation of the trust table.

The client can then determine whether or not to continue the handshake by performing the following checks:

- Verify that the certificate is valid at the current time.

- Hash the server certificate, *e.g.* using SHA-1, and check to see if it is in the trust table. If so, the handshake can proceed if the trust vector indicates that the certificate is an SSL server certificate. If the hash is in the table, but the SSL server bit is not set in the trust vector, the handshake immediately fails. If the hash is not in the table, proceed to the next check.

- If the CA's Root Certificate was not presented, retrieve it from the Security Service on the server.

- Verify the digital signature on the server certificate, and verify that the signature came from the named CA using the CA's Root Certificate.

- Hash the CA's Root Certificate, and check to see if it is in the trust table.

- Verify that the table entry for the CA's Root Certificate is marked as trusted as a CA for issuing SSL server certificates.

Once all these checks are complete, the client continues the SSL handshake by negotiating a cipher suite and exchanging keys with the server, and then begins transferring encrypted data.

Although the Liberate mechanism solves several problems, such approaches may have one or more of the following disadvantages:

- There are too many dependencies on the software provider. A software company may not be qualified to behave as a CA and perform certificate management. For example, if the software provider's root certificate is compromised then the whole system may be vulnerable.

- There is too much burden for the device owner and ESP. They must go through a series of steps to update their root certificates whenever their root certificates are expired or compromised. For example, the device owner must submit its new root certificate to the software provider and wait for a new RSIO that includes the new root certificate from the software provider. Then, the device owner must reconstruct a RSIO and deliver it to the ESP. The ESP must then reconstruct an RSO and deliver the updated information to the clients.

Such procedure is complicated. In the reality, the device owner and ESP are often the same party and, thus, the three-level hierarchy is redundant and creates overhead. Further, there are many human interactions involved whenever information in the RSIO must be updated. In such case, the turn around time may be too long.

It would be advantageous to provide a simplified trust information delivery scheme for certificate validation.

SUMMARY OF THE INVENTION

The invention provides a simplified trust information delivery scheme for certificate validation. The presently preferred embodiment of the invention comprises a server based certificate management mechanism that delivers certificates and associated trust information for clients to verify received certificates. Such scheme works effectively with SSL to provide server authentication. Such scheme may also be used for Javascript access control. The scheme especially fits into the consumer device environment.

In the preferred embodiment, a unique TIO based trust information delivery scheme allows clients to verify received certificates and control Javascript access efficiently. This scheme fits into the certificate verification process in SSL to provide a secure connection between a client and a Web server.

In particular, the scheme is well suited for integration into consumer devices that have a limited footprint, such as set-top boxes, cell phones, and handheld

computers. In such devices, it is not practical to hardcode a certificate database, which usually ranges from 50K to 400k in size, into the client memory due to limited amount of storage available in such consumer devices. Using the herein disclosed TIO technique, the size problem is substantially solved. Furthermore, the TIO update scheme allows clients, *i.e.* consumer devices, to update certificates securely and dynamically.

A key observation in connection with the invention herein is that in general, due to the limited resources such as memory and computing power, consumer devices need server support to browse a network, such as the Internet. The main function of such server is to reformat or transcode Web page contents so that the clients can display the results. Typically, transcoding servers are hosted by device owners, such as cable operators, ISPs, and broadcasters. The invention provides a mechanism by which these entities are the TIO providers that deliver the TIO to their client devices to enable the SSL functionality.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is an architectural block schematic diagram of a trust information delivery scheme for certificate validation according to the invention;

Fig. 2 is a flow diagram showing SSL authentication according to the invention;

Fig. 3 is a flow diagram showing Javascript access control according to the invention;

11

Fig. 4 is a flow diagram showing HTTP update according to the invention in a persistent storage client device;

Fig. 5 is a flow diagram showing broadcast update according to the invention in a persistent storage client device; and

Fig. 6 is a flow diagram showing client device update according to the invention in a device that does not include persistent storage.

DETAILED DESCRIPTION OF THE INVENTION

The presently preferred embodiment of the invention (see Fig. 1) comprises a scheme that allows Trust Information Providers (TIP) 10, *e.g.* cable operators, ISPs, and OEMs, to deliver certificates with associated trust information to clients 12 for verification of the received certificates. The scheme also allows the TIP to update the certificates and associated trust information sent to clients, and it operates in connection with both flash and non-flash based clients.

A key observation in connection with the invention herein is that in general, due to the limited resources such as memory and computing power, consumer devices need server support to browse a network, such as the Internet. The main function of such server is to reformat or transcode Web page contents so that the clients can display the results. Typically, transcoding servers are hosted by device owners, such as cable operators, ISPs, and broadcasters. The invention provides a mechanism by which these entities are the TIO providers that deliver a trust information object (TIO) 14 to their client devices to enable SSL functionality.

Trust Information Object (TIO)

Conceptually, a Trust Information Object (TIO) is a table of two columns
having a timestamp, the number of signatures and, optionally, digital
signatures. Each row of the table consists of the hash value of a trust entity
certificate, such as Root CA Certificate, and its associated trust information
indicating the level of the trust for this entity. Table "A" below illustrates an
exemplary structure of the TIO.

Table A. TIO Structure

HASH	TRUST VECTOR
$Hash(C_1)$	TV_1
$Hash(C_2)$	TV_2
...	...
$Hash(C_n)$	TV_n
Number of Signatures	
Timestamp	
Signatures	

13

TIO Structure

Where:

5   C_i – is a trusted entity's certificate, *e.g.* a CA root certificate or SSL

server certificate (for optimal performance, *i.e.* minimum amount of TIO

update and certificate fetching, the hash value can be taken on the

certificate excluding the validity and serial number);

10   TV_i – is the trust vector of certificate i;

Number of Signatures – specifies the number of signatures required

for the next update;

15   **Timestamp** – is the date the TIO is created; and

Signature – is the digital signature of all the data including the

Certificates, the Trust Vectors, the Number of Signatures, and the

Timestamp, contained in the TIO.

20

The preferred hash function can be either MD5 or SHA-1. For maximum

security, SHA-1 is presently preferred. The ASN.1 (see Abstract Syntax

Notation, ftp://ftp.rsa.com/pub/pkcs/ascii/layman.asc) definition of the TIO,

which follows the PKCS#7 standard (see The Public-Key Cryptography

25   Standards (PKCS), RSA Data Security, Inc., Version 1.5, revised Nov. 1,

1993, http://www.rsasecurity.com/rsalabs/pkcs/pkcs-7/), and the semantics of

14

each bit in the trust vector (TV) are described in greater detail below. Because the output of the hash function has a fixed length of twenty bytes maximum, *i.e.* when using SHA-1, and the TV is likely from one to two bytes, the size of the whole table is very small. Thus, the TIO readily fits into consumer devices, such as set-top boxes, cell phones, hand held computers, and pagers. For example, a TIO derived from 50 Root Certificates has the size of around 1k. Furthermore, with a TIO containing the hash values of the most popular root CA certificates, clients are capable of communicating with the majority of the secure web sites.

At the software development stage, a TIO derived from a set of popular root CA certificates is hard coded into the client software. In this embodiment, where the client, *i.e.* the consumer device, has flash memory 16, a copy of the TIO is saved in the flash memory during the client build time. The TIO is periodically updated thereafter using a mechanism described below.

SSL Server Authentication

During the SSL handshake (100; see Fig. 2) between the client and the server, the server sends a certificate chain that may or may not contain the root certificate (RC) to the client (102).

The client can validate the server certificate by the following procedure:

1.   The client hashes the server certificate (104) using SHA-1, assuming that SHA-1 is used in the construction of TIO, and compares the resulting

1 5

digest against the list of trusted entity certificate thumbprints obtained from the TIO (106). If a match is not found (108), step #2 below is performed. If a match is found (108), the client checks the trust bit vector associated with the certificate to ensure that the authenticated server is trusted in the context of

5   the SSL session being established (110). If the necessary trust capabilities are not set on the matched thumbprint (112), the client fails the SSL handshake (114). Otherwise, the server is deemed authenticated (116), provided that the remaining steps of the SSL handshake protocol are successfully completed.

10

2.   In the case where the chain does not contain the RC, the client first retrieves the RC from a trusted server through the normal HTTP operations (118). Without loss of generality, it is assumed that the RC is available in the client. Then the client goes through the normal certificate chain validation up

15 to the root CA (120). Once the entire chain is validated, the client tries to validate the CA RC (122). If the RC is included in the chain (124), then the client hashes the RC and looks up the TIO in the client (126). If the hash value and a corresponding trust bit (which indicates that the CA is trusted to issue SSL server certificates) are found in the TIO 108, 110, 112, then the

20 certificate chain is considered to be valid and the SSL handshake procedure proceeds 116. Otherwise, the certificate chain validation fails and the SSL negotiation stops 114.

CA root certificates have a finite life span and expire from time to time.
25 However, the expiry of a certificate does not imply that the certificate is

16

compromised and no longer can be used. Most CAs generate their new CA certificates using the old key pairs to minimize transition problems. In this situation, the old root certificates can still be trusted. To minimize the amount of CA fetching and TIO update, the hash value in the TIO can be taken by

5   hashing the certificate, excluding the validity and serial number. Doing so, the certificate described in the above SSL authentication process is accepted by the validation mechanism, even when the client receives an expired CA root certificate. This does not create a security hole as long as the TIO provider knows that the CA certificate is still valid.

10

Access Control

In general, client devices include a set of specific data, saved by the manufacturer or the service provider, that are accessible only to the trusted

15   applications. The update of these pieces of data is typically through Javascript or Java due to the mobility of these languages, although other languages can be used. In one embodiment (see Fig. 3), a designated trust bit with the site certificate in the TIO is used to identify a site that is trusted to perform special operations. When the client executes a Javascript thread (200) it checks the

20   certificate and associated trust information (202). If the trust bit indicates that the site identified by its certificate is trusted for the intended operation (204), then access permission is granted (206). Otherwise, the client rejects the access (208).

25   **Root Certificates Update**

17

A TIO that is hard coded in the client software or saved in the flash memory of the client device provides the trust basis for the client to make SSL connections. As is known, certificates may become invalid from time to time. Thus, a mechanism that allows TIO providers to update the TIO contained in

5   flash memory is necessary. The following discussion describes two different TIO update schemes: one of these schemes is suitable for clients having flash memory, *e.g.* NVRAM, while the other scheme is useful for clients that do not have any local persistent storage.

10   **Client with Flash/NVRAM**

In this case, the client device has a copy of the TIO in flash memory (see Fig. 1). In the preferred embodiment, the TIO is delivered to the client through one of two channels 18, *i.e.* broadcast or HTTP.

15

In the case of using HTTP over an SSL channel to deliver the TIO (see Fig. 4), the TIO does not need to be signed. Although it can also optionally be signed. At boot-up time 300, the client connects to the server 302 to ask whether a new TIO is available, and the server sends the new TIO 304 to the

20   client through the normal HTTP if there is a more recent TIO.

In the broadcast situation (see Fig. 5), the TIO must be signed. For the client to verify the signature of the TIO, the signing certificate of the authority, which may be a CA, software provider, or the cable operator, must be delivered to

25   the client before hand. To that end, the cable operator (for example) must send a TIO including the signing certificate to the client through the SSL

1 8

channel before using the broadcast method 400. The trust information of the signing certificate indicates that this certificate can be trusted for signing the TIO. The update procedure, depending on the mechanism (HTTP or broadcast), is illustrated in the following:

5

Update via HTTP

In the case where the client is capable of performing SSL, the client can fetch the TIO from the trusted server through HTTP over SSL, or via HTTPS. Because SSL guarantees the security of the delivery, the new TIO does not 10 need to be signed and thus no signature verification is needed. However, the client must ensure that the root certificate that signed the trusted server certificate is contained in the new TIO and not revocable, as indicated by the trust bit associated with the certificate. This check is performed to guarantee 15 that the system is always recoverable even when a malicious TIO hacks into the client. Otherwise, there is a potential situation that can cause the client device to fail. For example, if the server is compromised and a hacker manages to send a malicious TIO that contains only a malicious root certificate to the client, then the client can never connect to the server again 20 because the SSL server authentication fails. With a root certificate that signed the trusted server certificate that is not revocable, the client can always connect to the server through SSL to fetch a valid TIO and get rid of the malicious TIO. Then the client checks the timestamp that is embedded in the TIO. If it is valid, *e.g.* later in time than the previous TIO, then the old TIO is 25 replaced with the downloaded TIO. Otherwise, the client rejects the update request.

19

In the case where the TIO is digitally signed due to the requirement of a security policy or lack of support for SSL, the client verifies the digital signatures of the TIO with the signing certificates along with the TIO sent to the client. Here, multiple signatures may be verified, depending on the number of signatures specified in the TIO. Then the client hashes the signing certificates one by one. If the proper results are found in the TIO and the trust bits indicate that these certificates are trusted for signing TIO, then the TIO proves that it was not tampered with. Finally, the client verifies the timestamp in the same way as mentioned above.

Note that the signing certificates must exist in the TIO in the client before the TIO is signed. Otherwise, the signing certificates can not be validated. To that end, the TIO providers can either choose CAs whose root certificates are included in the initial TIO embedded in the client to sign the TIO, or they can use an SSL channel to deliver a TIO that contains the signing certificates to the client before signing the TIO.

Update via Broadcast

In many environments, such as cable television, centralized data can be delivered to end users by broadcast. The biggest advantage of broadcast is its efficiency. To take this advantage, the TIO can be delivered to clients by broadcast. In this case, the TIO must be signed because the broadcast channel is usually not secure. The client can verify the TIO by the same procedure discussed above. The TIO providers deliver a TIO that contains the

signing certificate and associated trust information to the client by either including the signing certificate in the initial TIO saved in the client flash, or by sending the TIO to the client through the SSL channel before using the broadcast channel.

5

Client without Persistent Storage

In this situation (see Fig. 6), the same TIO update mechanism, *e.g.* HTTP/HTTPS channel or broadcast, as discussed above can be applied.
10 However, the update must occur on a per session basis because the TIO can only be cached in the device memory, *i.e.* it is not persistently stored. This means that, at each boot-up time 500, the client must fetch the most recent TIO from the server through the SSL channel 502 and receive any update thereafter from the broadcast channel 504. If a root certificate in the client TIO
15 is compromised or insecure, a software update can not be avoided. Because the chance of a root CA certificate being insecure is very small (it may not even happen at all; at least, there is no such evidence so far after more than five years of CA practice in the industry), the concern for the frequency of the software update can be ignored.

20

Note that expiration of a root CA certificate does not require updating of the TIO because the expiry of a root certificate only creates robustness problems, but not security implications if the key of the certificate known to be valid. The fact that the TIO update is not needed even when a root CA certificate in the
25 TIO is expired is true only if the renewed CA certificate shares the same public key with the old one. That is, the renewal differs from the old one only

by validity. If the renewal changed the key then TIO update is needed. The TIO does not need to be updated when the new CA certificate shares the same key with the old one because of the nature of the public key cryptography. If the new CA certificate shares the same key with the old one,

5 then the server certificate issued by the new CA can be cryptographically verified by the new CA certificate if and only if it can be verified cryptographically by the old CA certificate. That means the new CA certificate plays the same role as the old one in verification of the server certificate.

10 The only situation that the expiry of root certificates causes a software update is when all the root certificates in the TIO are expired and their corresponding key pairs are changed. This is because the SSL session between the trusted server that provides the TIO and the client can not be established. However, the chance for this to happen is very small because a new software release is likely to happen before all of the root certificates have expired. Thus, a cable

15 operator (for example) can always choose a CA known to be valid in the TIO to issue its trust server certificate, and the SSL session between the client and the trust server can be established all the time. Once the SSL session between the client and the server is established, the client can fetch the most

20 recent TIO, which contains only the valid root certificates, from the server and updates the old one with the same procedure described above. The new TIO is then cached in the memory for the subsequent SSL session establishments.

22

ASN.1 Definition of Trust Information Object (TIO)

In the presently preferred embodiment of the invention, when the TIO is
signed, it is implemented using the PKCS #7 data format with the SignedData
5   encapsulation format. Table "B" below describes how the SignedData content
type is used for this purpose.

Table B. Use of the Signed Data Format to Implement the TIO

PKCS #7 Field Name	Description
SignedData.version	The PKCS #7 standard version – shall specify version 1.
SignedData.digestAlgorithms	The digest algorithms -- shall specify SHA-1 and, optionally, others.
SignedData.contentInfo.contentType	The type of data being signed – shall specify Trust-Info, as defined below.
SignedData.contentInfo.content	The data being signed -- shall contain the DER encoding of the ASN.1 object TrustInfo, as defined below.
SignedData.certificates	This is the list of root certificate fingerprints and associated trust information.
SignedData.crls	Not used (omitted).
SignedData.signerInfos	The version number for the digital signature format-- shall specify version 1.
SignedData.signerInfos.authenticated Attribute	Not used (omitted).
SignedData.signerInfos.digestEncrypti onAlgorithm	The digital signature algorithm – shall specify RSA.
SignedData.signerInfos.encryptedDig est	The digital signature.
SignedData.signerInfos.unauthenticat	Not used (omitted).

23

edAttributes	

The following is the ASN.1 definition for the Trust Information Object which is DER encoded into the SignedData.contentInfo.content field of the PKCS#7 SignedData object described in Table "B."

```
            TrustInfoObject ::= SEQUENCE
    {       timeStamp    UTCTime
    trustInfo      TrustInfo
    }


    TrustInfo ::= SET OF TrustEntity
```

timeStamp: The date the TIO is generated. This is a coordinated universal time or Greenwich Mean Time (GMT) value.

```
    TrustEntity ::= SEQUENCE
    {
            trustAttribute       CAType
            thumbPrint           BIT STRING
    }
```

trustAttribute: The trust information associated with an entity represented by its certificate.

thumbPrint: The SHA-1 hash of the public key embedded in the certificate that represents a trusted entity.

CAType ::= BIT STRING

{

    TIOCA   (0),   -- CA trusted to sign the TIO. This bit is used in the verification of a signed TIO. The client checks whether this particular bit in the trust vector is turned on when the TIO signing certificate hash is found in the TIO –

    SslClientCA   (1),   -- CA trusted to issue certs for SSL clients. This bit is used to authenticate server certificates in SSL. The client checks whether this particular bit in the trust vector associated with the root CA certificate that signed the SSL server certificate is turned on –

    SslServerCA   (2),   -- CA trusted to issue certs for SSL servers

–

    SslServerCert (3)   -- SSL server certificate, this bit indicates whether a received server certificate in SSL can be trusted as a SSL server certificate --

    appSwPubCA   (4),   -- CA trusted to issue certificates for Application Software Publishers. This bit is used for software update. When a new version of software, either signed or from a secure server (SSL server), delivered to the client, the client checks this bit to see whether the associated certificate is trusted for the intended access. –

privAppCA (5) -- CA trusted to issue certificates for Privileged Application Provider. This bit is for Java and/or Javascript access control. If this bit is turned on for a root CA certificate then any downloaded applications, signed or over SSL, whose associated certificates are signed by this CA are qualified to perform pre-defined privileged operations. --

}

Note that the trust bits can be added or removed depending on the requirements. For example, a TIO provider may use a few bits to support multiple levels of Javascript/Java access control. In this case, each bit identifies a privilege level. The presently preferred embodiment of the invention uses only one bit, namely privAppCA for this purpose.

Although the invention is described herein with reference to the preferred embodiment, one skilled in the art will readily appreciate that other applications may be substituted for those set forth herein without departing from the spirit and scope of the present invention. Accordingly, the invention should only be limited by the Claims included below.